MULTI-AGENT SYSTEMS IN COMMUNICATIONS

AFOSR RESEARCH PROJECT

project contract no.: F61775-99-WE099

Report No.: 3

Saving Communication in Multi-Agent Systems with Tri-base Acquaintance Model and Machine Learning

---

Michal Pěchouček, Olga Štěpánková, Vladimír Mařík

Gerstner Laboratory for Intelligent Decision Making and Control,
Department of Cybernetics,
Czech Technical University in Prague

REPORT DOCUMENTATION PAGE

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (FROM - TO)
26-09-2000	Final Report	xx-xx-2000 to xx-xx-2000

4. TITLE AND SUBTITLE Applications of Multi-Agent Systems in Communications Unclassified	**5a. CONTRACT NUMBER** F61775-99-WE099
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Pechoucek, Michal ; Stepankova, Olga ; Marik, Vladimir ;	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME AND ADDRESS Czech Technical University of Prague Department of Cybernetics (K333) CTU FEL Technicka 2 Prague CZ 166 27 Praha 6 , Czech Republic 166 27	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME AND ADDRESS EOARD PSC 802 BOX 14 FPO , 09499-0200	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) SPC 99-4099

12. DISTRIBUTION/AVAILABILITY STATEMENT
A
PUBLIC RELEASE

EOARD
PSC 802 BOX 14

FPO , 09499-0200

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report results from a contract tasking Czech Technical University of Prague as follows: The contractor will investigate and further develop the applications multi-agent systems can play within modern information and communication systems. Specifically, the contractor will investigate unique capabilities: data-mining techniques to enhance minimized communication between software-agents, as well as the capability to detect variances in transmitted data. The contractor will document these developments in two interim reports, as well as deliver a conclusive final report and data models as detailed in the technical proposal.

15. SUBJECT TERMS

EOARD; Agent Based systems; defensive information warfare

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
					Fenster, Lynn lfenster@dtic.mil
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	Public Release	35	**19b. TELEPHONE NUMBER** **International Area Code** **Area Code Telephone Number** 703 767-9007 **DSN** 427-9007

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	26-September-2000	Final Report

4. TITLE AND SUBTITLE

Applications of Multi-Agent Systems in Communications

5. FUNDING NUMBERS

F61775-99-WE099

6. AUTHOR(S)

Michal Pechoucek, Olga Stepankova, Professor Vladimir Marik

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Czech Technical University of Prague
Department of Cybernetics (K333) CTU FEL
Technická 2,
Prague CZ 166 27 Praha 6
Czech Republic

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
PSC 802 BOX 14
FPO 09499-0200

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

SPC 99-4099

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

A

13. ABSTRACT (Maximum 200 words)

This report results from a contract tasking Czech Technical University of Prague as follows: The contractor will investigate and further develop the applications multi-agent systems can play within modern information and communication systems. Specifically, the contractor will investigate unique capabilities: data-mining techniques to enhance minimized communication between software-agents, as well as the capability to detect variances in transmitted data. The contractor will document these developments in two interim reports, as well as deliver a conclusive final report and data models as detailed in the technical proposal.

14. SUBJECT TERMS

EOARD, Agent Based Systems, Defensive Information Warfare

15. NUMBER OF PAGES

35 plus electronic copies of model documentation files

16. PRICE CODE

N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19, SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Content

Introduction

This report summaries results of the research carried out within the last phase of the project contract no.: F61775-99-WE099 - Multi-Agent Systems in Communications (MASiC). Results outlined in the report are accompanied by a multi-agent system prototype that was a subject of the testing.

MASiC research effort aimed at identifying mechanisms for saving communication traffic in multi-agent systems. Within the first phase of the research project [Pechoucek 00b] we have designed a tri-base acquaintance model (3bA) as a formal model of agent's mutual awareness. Within the second phase [Pechoucek 00c] we have implemented an agent with a 3bA model (tri-base agent) that has been incorporated within the ProPlanT – multi-agent system. The last phase has been devoted to experimenting with the system, justifying the proposed methodology and suggesting spots where the overall performance of the tri-base acquaintance model may be improved by machine learning techniques.

This report starts with brief summary on the philosophy of the 3bA model and discussion of its contribution in communication savings in a multi-agent system. After that we show experimental results we have achieved on testing a sample community. The report concludes with a study of possible utilization of machine learning techniques in communication efficiency improvements. The appendix provides a user documentation to the experimental demonstration Java-based 3bA agent within the ProPlanT multi-agent environment.

Acquaintance Model Communication Savings

This report is a part of the overall MASiC project documentation. The MASiC phase 1 report [Pechoucek 00b] is a prerequisite for thorough understanding of this study. The Tri-base Acquaintance Model (3bA) is a formal model of agents' mutual awareness [Marik 99a], for the review of the used notation see Appendix. It provides the agent with knowledge structures and mechanisms of maintaining social knowledge about other collaborating agents in order to avoid expensive communication. Precompiled social knowledge used for fast, efficient and accurate decision making is administered in three separate bases:

- Cooperator base – maintains permanent knowledge about collaborating members of the community (e.g. their abilities, addresses, communication languages, etc.).
- State base – maintains nonpermanent information about actual load and agendas of collaborating agents. This base comprises two sections: Task Section – collecting information about the course of planning and executing of the processed task and Agent Section – containing information about the actual load of the collaborating agents.
- Task base – contains pre-prepared plans how to decompose and delegate possible requests. This base has been also broken into two sections: Problem Section (PRS) – which is a set of task-decomposition rules and Plan Section (PLS) – containing a complete, close to optimal, decomposition plans.

As we will demonstrate throughout this study the tri-base acquaintance model saves substantial portion of communication requirements of the multi-agent system. The communication efficiency improvements provided by this approach may be seen as a specific communication load shift. The communication load is minimized in the agent's *critical time* (i.e.: moment when the agents are required to fulfill a request) while a new model maintenance activity appears in the agent's *idle time*. Each request is answered quickly but it brings substantial communication flows after answering the request.

Hereafter we will refer to a contractor as an agent who contracts another agent with a request. A contractee is an agent who was contracted by another agent with respect to a request. Similarly a subscriber is an agent who subscribes another agents for reporting on its services and a subscribee is an agent who was subscribed. Subscribee keeps sending advertises to a subscriber each time its status gets updated [Pechoucek 00].

Methods of communication

In order to justify the proposed methodology and provide solid experimental results, we comparethe load of the acquaintance model type of communication with simple broadcasting mechanism. We will illustrate results of our work on a very typical communication stereotype – a *request for decomposition*. If an agent is requested to decompose a task it shall detect the best possible collaborators (based on its knowledge of decomposition) and contract these with parts of the original request.

Hereafter we will refer to *broadcasting based task decomposition* as a communication convention consisting of three phases – (i) broadcast phase where a decomposing agent (*contractee)* broadcasts a tender for cooperation, (ii) bidding phase where interested agents reply their offers and (iii) contract phase when the contractee selects the best possible contractor and contracts it with a request for collaboration. We will refer to *acquaintance model based decomposition* as a communication protocol that

involves two phases – (i) contract phase when the contractee contracts best possible agents for collaboration (decision is based on knowledge stored in the agent's acquaintance model) and (ii) model maintenance phase when contracted agent advertises change of its status (caused by the contract) within the group of its subscribers, i.e. agents who subscribed the appropriate agent for this information. Developers in the team used terminology based on KQML performatives corresponding to each of the phases. The contract phase is referred to as an achieve phase, the broadcast and the bidding phases are jointly named as evaluate phase and the model maintenance phase is called advertise phase. There is another quite important communication phase in the life of the community, which we call register phase. In this phase agents get constructed and mutually acquainted. This process is inevitable for further utilization of an acquaintance model. Naturally the process of mutual exchange of information among the agents is expensive in terms of communication and may become a bottleneck in the case of large communities.

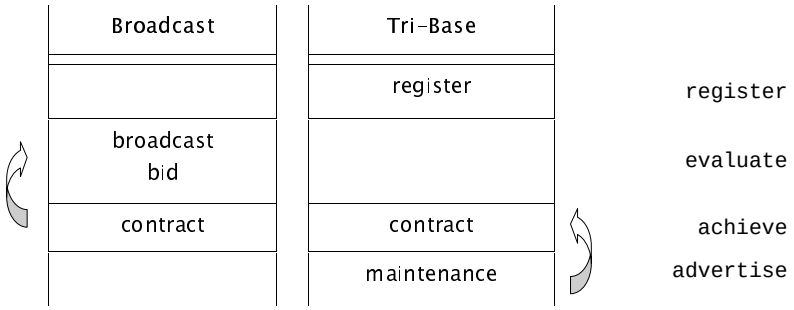


Figure 1 – Models of Communication

In the report we will refer to a broadcasting agent as an agent who does not maintain an acquaintance model and carries out broadcasting based task decomposition and to acquaintance based agent as an agent who decomposes requirements by using social knowledge stored in its acquaintance model and thus carries out acquaintance model based decomposition

Let us consider an acquaintance based agent A to be free to decide about its collaborators if its knowledge base (PRS in the TB) provides several alternatives how to decompose at least one requirement and how to delegate responsibility:

$$A \text{ is free iff } \exists T \in \beta(A)\ (|\omega(A,T)| > 1),$$

where $|M|$ denotes the cardinality of the set M.

A *degree of cooperation freedom* of the community Θ, denoted as $\varphi(\Theta)$, is the number of agents who are free to make their choice whom to contract and whom to delegate with decomposed subtasks:

$$\varphi(\Theta) = |\{ A: A \in \Theta \wedge A \text{ is free}\}|$$

Any free agent either broadcasts a request and waits for offers from possible candidates or uses some piece of social knowledge to make the rational decision. On the other hand, if agent's knowledge provides only fixed decomposition assignments, the agent does not have any freedom to select its collaborators. It is obvious that in the latter case the communication complexity is substantially smaller. The degree of cooperation freedom of the community is thus equivalent to number of agents free to form the team collaborators for a specific project.

However for many purposes the introduced metric is not precise enough. For each particular request the community has different degree of cooperation freedom (as a specific agent can choose between two variants when solving certain task, but for another task there is only one decomposition

possible). We say, that the agent *B* is free for the task *r* iff *B* knows about more than one decompositions for *r*, i.e.

$$\text{agent } B \text{ is free for the task } r \text{ iff } |\omega(B, r)| > 1.$$

Let $\varphi_r(\Theta)$ denote the number of agents which are free for the task *r*. Then $\varphi(\Theta)$ as defined above is an upper bound estimate of $\varphi_r(\Theta)$ for each particular task *r*. A more precise metric may be the sum of all degrees $\varphi_r(\Theta)$ per each request *r*,

$$\sum_{r \in S} \varphi_r(\Theta),$$

or the average degree of freedom per single request.

Complexity of the decision making process is significantly influenced by "substitutability" among agents. Thus another interesting measure corresponds to the maximal number of different agents, which are able to solve a single task. In the rest of the report we will stick to the first measure $\varphi(\Theta)$, refered to as the degree of cooperating freedom.

Communities Tested

Utilization of the 3bA model has been experimentally justified by means of tests carried out in the ProPlanT multi-agent system. As explained in the first report, there are three fundamental classes of the system's community. For the purpose of further explanation let us distinguish among decomposition agents (DA), where we include ProPlanT PPA (production planning agents) and PMA (production managing agents) and resource agents (RA), where we count PA (production agents) agents of various types. While resource agents provide other agents with some service or commodity, the decomposition agents are responsible for optimal decomposition of a task into a set of subtasks and further delegation among collaborating agents. This classification of agents in terms of their capability to decompose and delegate bears a resemblance to the Pleiades architecture of collaborative agents architecture that consists of *task-specific* agents (TA) and *information-specific* agents (IA) [Sycara 95].

As we did not have an access to military-related test cases and experimental data, entire simulation was carried out on the community of agents planning manufacturing of compressors within a factory. The task of the community is to come to a general agreement on how to allocate given pieces of production when considering an internal structure of the manufacturing process.

We have experimented with three different configurations of the "same" community, i.e. a community with the same functionality (set of treated tasks) as the original one. The original community consists of 17 agents of a different nature, 6 of which are free (have a free choice to contract the other agents and thus decide on optimality). Moreover, for any free agent holds that all the tasks he treats have at least 2 different decompositions.

As already noted the functionality of the multi-agent system (*Figure 2*) has been implemented by means of three different community configurations. Each new configuration is derived from the original one by adding (eventually simplified) "copies" of selected agents existing in the original community:

- community no. 1 – The original community consisting of 6 decomposition agents (management, quality, production, dispatching, materiel and pca) and 11 RA (coop1, coop2, store,

ass1, ass2, works1, works2, tester, road, train, config). This is the original community has 17 agents[1].

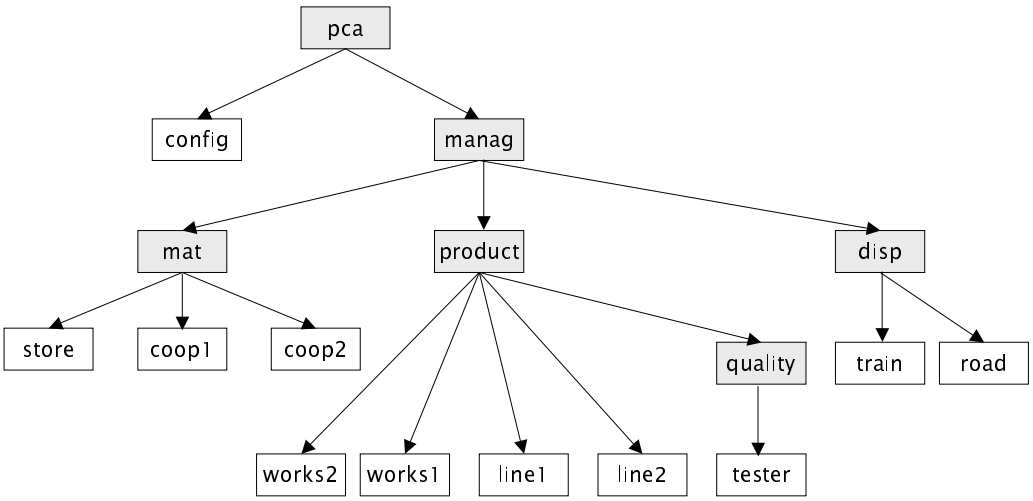


Figure 2 – Testing Community no. 1.

- community no. 2 – Higher communication traffic has been tested on this community. We have extended the community with several other DA agents. As the overall functionality of the community shall not be modified, separate DA agent has been constructed for each single task, which any of decomposition agents from the first community accounts for. This is why the number of decomposition agents has risen up to 36 (pca 1, management 9, quality 5, production 9, dispatching 3, material 9), while all of them remained free. The total number of agents for this case is 47 and the degree of freedom is 36.
- community no. 3 – The community with even higher communication traffic has been created from the community no. 2 by separating the resource agents so that there is a single RA accountable for a single task. By doing so we got 78 RA agents (coop1 12, coop2 12, store 11, ass1 9, ass2 12, works1 6, works2 9, tester 5, road 2, train 2, config 1). Total number of agents is thus 115 while the degree of freedom still remained 36.

Experimental Results

In order to confirm the communication savings estimates we have designed two different experiments. Firstly communication requirements of the tri-base agent and broadcasting agent were compared with respect to different degree of freedom of a single community – community no. 1. Within this experiment we modified the original community in order to achieve various modifications with a different degree of freedom. Modifications of the degree of freedom resulted from changes of different agents' knowledge stored in their task bases. Within the second experiment we measured communication savings in three different communities of agents.

[1] Note that DA agents are represented by boxes with gray fillings while RA are represented by boxes with white fillings.

Broadcasting Agent vrs. Tri-base Agent with Different Degree of Freedom

We have carried out simple tests in order to compare communication requirements for a single decomposition request sent to the community of broadcasting agents and tri-base agents with different degrees of cooperation freedom. This experiment has been carried out in the simplest community from above – community no. 1. The degree of cooperation freedom has been set by interventions into agents' decomposition knowledge stored in their knowledge bases by disabling or adding some alternative decompositions for specific tasks. We have experimented with both the marginal cases – no cooperation freedom (community no. 1a) and full cooperation freedom of all decomposition agents (community no. **1**), as well as with the case of only two free agents (community no. 1b). The last community is very similar to our real life example form production planning. For results see *Table 1*.

community	φ(Θ)	number of messages		
		m(3bA)	m(broadcast)	m(maint)
1a	0	30	30	123
1b	2	30	46	123
1	7	30	180	123

Table 1 – Test no. 1 results

The first column of the *Table 1* (m(3bA)) shows number of messages sent within the community during the achieve phase. As the degree of freedom affects communication only in the evaluate (in the case of broadcasting), there is still the same number of messages measured for each $\varphi(\Theta)$. Although communication in the maintenance phase – the third column of the table – does not depend on $\varphi(\Theta)$. It is rather related to the architecture of the community, as will be illustrated in the next experiment. Therefore there is the same value in each row. The second column gives the total number of messages sent by the broadcasting agent. The first value is the same as in the tri-base agent: for $\varphi(\Theta)=0$ there is no decision making process about whom to delegate the decomposition with and therefore there is no evaluate phase. The last number illustrates the case where each agent who is contracted with a request broadcasts a call for cooperation within all agents, who consequently reply with their offers and the agent contracts the optimal one.

The higher is the degree of cooperation freedom of the community, the more communication traffic is required and the acquaintance model rises its impact on the communication savings. In the marginal case with fixed task decomposition and job delegation ($\varphi(\Theta) = 0$), the acquaintance model has no utilization while substantial communication needed for the model maintenance is required. The broadcasting agent does not have to carry out any bidding (evaluate communication phase) and directly contracts (achieve communication phase) the single relevant agent (of which the broadcasting agent knows from the communication structure). Broadcasting based decomposition consists in this case from the achieve stage communication only and therefore communication requirements for either approaches shall be identical (m(3bA) = m(broadcast) = 30). However, the tri-base agent needs to spend certain amount of communication resources for maintenance of their models (m(maintenance) = 123). This is why utilization of the tri-base models is not very likely to pay off in this case. On the other hand, in the case of total cooperation freedom ($\varphi(\Theta) = 7$) in the considered community no. 1, the broadcasting agent would have to exchange six times more messages (m(broadcast) = 180) than the tri-base agent (m(tri-base) = 30) in the community critical time. The total number of messages sent in the whole communication cycle of tri-base agent is also smaller (m(tri-base)+m(maintenance) = 153 < m(broadcast) = 180). In the mentioned example, reflecting the

real organizational structure in the plant (see [Pechoucek 00a]), there are only two agents (production, material) making their own choices in terms of optimal allocation of collaborating agents workload. An interesting observation is that in this case with rather low degree of collaboration freedom ($\varphi(\Theta)$ = 2) the tri-base agent saves already more than one third of communication requirements (m(tri-base) = 30 < m(broadcast) = 46) in the critical times.

Broadcasting Agent vrs. Tri-base Agent in Different Communties

The principal subject of the second experiment was the community of 17 agents with seven free agents ($\varphi(\Theta)$ = 7) – community no.1. Communication savings were tested also in other two communities – community no. 2 and community no. 3, whose architectures were based on functionality of the community no. 1. Although with increasing number of agents the degree of freedom of the community may increase[2], for a specific request there are still two free agents within the available part of the involved community. This is why the degree of freedom was not an issue when experimenting with the different communities.

Table 2 summarizes the test results. The first part of the table provides detailed structure of each of communities. Second part of the table gives a full amount of communication messages per decomposition cycle for acquaintance based decomposition. Each row gives number of messages in an appropriate communication phase of one decomposition cycle. In addition to this we illustrate communication requirement in the registration phase. The last part of the table provides results of measurements of the broadcasting based decomposition communication requirements.

	community no. 1	community no. 2	community no. 3
number of RA agents	11	11	78
number of DA agents	6	36	36
total number of agents	17	47	115

	acquaintance based decomposition - number of messages		
register	173	1871	4517
contract	30	30	30
maintenance	123	206	31
total per request	153	236	61

	broadcasting based decomposition - number of messages		
register	27	58	196
evaluate	150	285	788
achieve	30	30	30
total per request	183	315	818

Table 2 – Test no. 2 results

[2] $\varphi(\Theta)$ increases with increasing number of agents in our case because new DA agents were constructed so that each DA agent accounts for one particular task, while still maintaining the same number of decomposition alternatives as in the community no. 1. If a new community was constructed so some DA agent there would be constructed agents where a single agent would account for one decomposition alternatives, the decision making freedom would be shifted up to the agent who contracts the agent in question and the degree of freedom may eventually decrease. This is not however our case.

As expected, experiments show that with increasing number of agents in the community the 3bA model provides more of communication savings. The overall degree of freedom rises with increasing number of agents. Consequently this observation is not in contradiction with results from the previous experiments.

Another important observation regards sharing resources. Sharing resources brings difficulties to the tri-base agents. While the community of 47 agents needs 206 messages for model maintenance 115 agents community requires 31 maintenance messages. The reason for this is that in the former community no. 2 there are 36 DA agents contracting 11 RA agents. Resource agents have to be therefore inevitably shared. The latter community no. 3 is organized in a tree and this is why no RA agent may be contracted by two different DAs and thus it advertises to single agent only. However, the organization of the community does not have any impact on communication load of the broadcasting agent. The increased number of RA agents will economize the flow of advertise messages sent from the level of RAs towards DA agents. As the community is organized in tree-like manner with the high branching factor in the close-to-leaves levels of the tree, it is easy to back-propagate an update within the community (maintenance – 31) than searching for an optimal configuration (evaluate – 788).

The drawback of all the acquaintance-model-based approaches is the register phase. Unlike broadcasting agent, the tri-base agent has to set subscribe-advertise links with its pears prior to functioning of the community. This is what is rather time consuming and may become a bottleneck with large communities (4517 messages among 115 agents). Fortunately this process has to be gone through only once when the community is created. However, new ideas how to optimize this problem are discussed later within the report.

Frequency

As already mentioned the communication savings offered by this approach may be seen as a specific communication load shift. Tri-base minimizes communication load in the agent's contract phases while a new advertising activity appears in the agent's maintenance phase. Each request is answered quickly but it brings substantial communication flows following the request fulfillment. The contract phase is usually placed in community *critical time* of operation, when agents are required to answer promptly. The virtue is that the maintenance phase is carried out in the community *idle times*, where agents are not required to provide any assistance to the user. Successful operation of such a multi-agent system depends on the community lifecycle. In order to utilize this acquaintance model based mechanism and to guarantee its communication savings for the frequency *f* of requirements to plan the following condition should be met

$$\frac{1}{f} \geq t_c + t_i ,$$

where t_c is maximum amount of time spent in the *critical time planning* and t_i is maximum amount of time needed for processing the subscription/advertise mechanism in agents' *idle time*.

This observation suggests exclusion of possible processing of parallel requests within the community (as minimal gap between two consecutive requests is required). If there are two completely independent requests coming at the same time, quality of the decomposition, time of the response and communication requirements are unaffected by this irregularity.

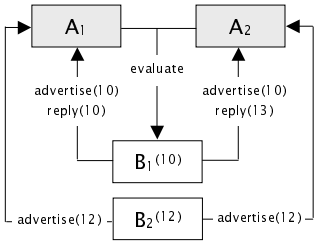


Figure 3 – Conflicts caused by shared resources

However sharing resources causes conflicts. Let us assume that two different requests will need to share resources of a single agent. In the simplest possible case of the previous two agents (A_1, A_2) try to contract either of their two subscribees (B_1, B_2) for carrying out task t (time(t)=3). Each of subsribees advertised its offer – $B_1(t) = 10$ and $B_2(t) = 12$. The agent A_1 makes a sound decision and contracts agent B_1 with a task t. Provided the agent A_2, who did the same decision as the agent A_1, contracts the agent B_1 prior to agent B_1 advertising a new offer (thus failing to meet the above mentioned condition) A_2 will obtain a reply that will be in conflict with the social information the agent A_2 maintains about the agent B_1 in its wrapper.

In the current implementation of the tri-base agent within the community fails to provide an optimal answer due to the conflict mentioned above and brings about a global failure (sorry KQML message). In another philosophy the community will ignore the conflict and will reply with an answer that will not be optimal (such as agent A_2 accepting collaboration with B_1 although the task will be completed in 13 times unit where the optimal solution would be provided by agent B_2 in 12 time units). The best possible approach suggests refusing such a contract, updating the tri-base model according to conflict and carry out local re-planning. It is easy to understand that this approach brings an optimal answer but at the cost of some extra communication. This is what we wanted to emphasize in this paragraph.

Open Issues

When applying the tri-base acquaintance model in critical domains one would need to economize several aspects of the knowledge maintenance mechanisms.

Register-Phase

Firstly (and maybe most importantly) the communication load in the register-phase of the community shall be brought to minimum. We have carried out separate research in the areas of efficient community configuration. The currently investigated ideas are based on replacing the presently used mechanisms of configuring communication links. These mechanisms are based on an intense broadcast within the entire community, where each decomposition agent is fishing for possible future collaborators.

An elegant solution is to utilize a facilitator, who the agents have to register with if they desire to participate in the community. Apart from their symbolic name, IP address and port number, the agents will register also their capabilities and services. Such a community component will resemble functionality of a FIPA [fipa98] normative directory facilitator (DF) who provides "yellow pages" services to other agents. Even though this is a standard way, we wanted to test our approach in the environment with minimal centrality and maximum autonomous rationality of participating agents.

Advertise Messages

Another open issue regards optimization of the advertise messages sent after an update of the community. Currently we send a lot of unneeded information when a single little change is advertised to subscribers. Firstly we have to decide what is a relevant change. Combination of (i) how old is the latest advertise, (ii) how relevant is the information to be advertised and (iii) how substantial is the advertised change shall be considered when triggering knowledge maintenance process through advertise messages. We came across this problem when implementing real time reasoning into our agents. Each tick of a timer triggers a massive re-computation of all advertise messages and causes communication overload.

Cooperation Neighborhood

In order to optimize the communication flow we have to maintain the agents' cooperation neighborhood $\varepsilon^t(A)$. We have shown how the agents can join someone's cooperation neighborhood. However neither means for subscribers to leave the cooperation neighborhood, nor for subscribee to discontinue advertising to a subscriber, have been until now implemented and experimented with. For comments on this issue see the next section.

Problem Solving Neighborhood

Another optimization challenge of the community of planning agents is based on careful maintenance of the agent's problem solving neighborhood $\pi^t(A)$. An agent may be constructed with predefined responsibility to account for certain task, which is rarely requested. It hardly pays off to maintain such a decomposition plan pre-prepared in the agent's task base. It is rather uneasy to decide which task is important and which shall not be kept within the agent's problem solving neighborhood.

Machine Learning in 3bA Model Optimization

Though the experimental results showed substantial savings of the overall communication traffic (provided given assumptions), full exploitation of this methodology is based on proper maintenance of it's the 3bA content but also with dynamics of its structure. The previous section indicates that the agent's acquaintance model shall not be only a static container of data maintained primarily by collaborating peer agents. Agent itself shall use its experience in order to affect many aspects of the knowledge maintenance process dynamics.

Knowledge acquired from the historical behaviour of the system or from a part of the system contributes to overall systems efficiency improvements. In principle, we distinguish between

- single agent learning, where one individual agent analyses accessible data (most likely record of agent's historical performance) in order to revise knowledge stored in its tri-base
- meta-agent learning, where the overall community behaviour is analyzed and efficiency improvements are sent to individual agents who update their social models.

For the complete list of efficiency improvements caused by possible tri-base revisions see [Stepankova 98]

Single Agent Learning

Cooperation Neighborhood Optimization

As defined [Pechoucek 00], the agents' cooperation neighborhood $\varepsilon^t(A)$ is a set of agent's A collaborators at the time instant t. More precisely we talk about agents, the agent A subscribed for reporting on their actual status. These agents are listed in the *agent section* (AS) of the agent's *state base* (SB).

In the previous section we said that although members of the ProPlanT community can join someone's cooperation neighborhood, there are no means for subscribers to restrict the cooperation neighborhood, or for subscribee to discontinue advertising to a subscriber. Introduction of a new performative – unsubscribe – will allow subscribers to restrict their cooperation neighborhoods as well as the performative subscribe expands it. Subscribees do not necessarily need to be benevolent. Subscribed agents may easily discontinue advertising their status in spite of the fact they are subscribed. They have to have only a good reason for doing so.

We have shown that implementing technical means for maintenance of the agents' cooperation neighborhood is not a big research challenge. What is much more interesting, is detection of when and how the cooperation neighborhood would be revised. Let us call a subscriber-subscribee link to be unexploited if and only if subscriber did not contract the subscribee for certain threshold period of time (t_{old}). Restriction of the cooperation neighborhood may be thus driven two-fold:

- either the subscriber agent may analyze record of its communication history and easily detect unexploited links (with respect to t_{old}) and unsubscribe appropriate subscribees,
- or the sunscribee agent will analyze its record of past communication and stop advertising to agents who did not contract it for longer than t_{old} (the subscriber shall be informed of this).

If the subscriber needs to contract unsubscribed subscribee, it subscribes it first and then does usual decision making. The communication requirements of this process will not be worse than local broadcasting communication. This is why, careful specification of the threshold value t_{old} will affect the overall system efficiency.

It is very often the case that a single agent advertises lot of unneeded information though just information about one particular resource is looked-for. If we assume that a subscriber subscribes/unsubscribes not only an agent but an agent for a particular task, the identical mechanism may be used with advantage for optimization of subscribed information the agents advertise.

In this way the community will start with the biggest possible cooperation neighborhood and will restrict it throughout the life of the community. An alternative way is to start with all links unsubscribed and subscribe a new link when a requirement to contract arises. Experiments (which were not part of this study) may disclose which of these two approaches provides better overall system efficiency and indicate what is the relationship between the threshold value t_{old} and amount of communication within the system, a measure of the promptness of response and system's efficiency as such.

Amount of advertised messages can be also pruned with respect to how relevant is the reported update. Let us assume that the state change of the agent A is relevant if either its load changes so that

$$\frac{\left|\mathsf{Load}_1(A) - \mathsf{Load}_2(A)\right|}{\mathsf{Load}_1(A)} \geq t_{rel}$$

or there is at least one $\langle T, \mathsf{Cost}(T)\rangle \in \mathsf{Cap}(B)$ for which holds

$$\frac{\left|\mathsf{Cost}_1(A) - \mathsf{Cost}_2(A)\right|}{\mathsf{Cost}_1(A)} \geq t_{rel}.$$

In other words we say that any advertise update is relevant if a change of at least on a piece of advertised information is bigger than t_{rel}. Though it does not have anything much to do with cooperation neighborhood, in the sense as defined within the project, avoiding irrelevant update report may save substantial part of communication.

Another way how we can reduce the amount of communicated messages in the knowledge maintenance process is based on periodical revisions carried out by the community [Cao 97][Stepankova 98]. If the frequency of requests is much smaller than agents' states updates, there is a large number of advertise messages that no one actually needed. So instead of the state update back propagation in the moment of the slightest, unimportant change of a single RA agent within community, agents advertise in certain periodical moments. We did not consider this way as very promising. In order to save communication, periodical revisions would have to be comparatively rare. This worsens average precision of the agents' social model (imprecise social model causes another communication requirements in the moment of request). Increasing frequency of periodical revisions improves this precision but will increase communication requirements that will be close to the classical subscribe-advertise mechanism. Precise comparison would have to be grounded in experiments.

Problem Solving Neighborhood Optimization

While in the previous paragraph we discussed optimization of subscription links and corresponding communication flows, here we will briefly comment optimization of problem solving resources of an individual agent. The subject of our discussion will be the amount of plans stored in the plan section (PlS) of the agent's task base (TB) and the question how often this structure has to be revised.

According to [Pechoucek 00b] each change of the agents state base (SB) will inevitably trigger a revision of the PlS. A change in a load of a subscribed agent will affect all plans in PlS that rely upon

the subscribed agent. These plans shall be revised and the entire structure reordered. With limited agent's cooperation neighborhood we reduce the number of events that cause PIS revision.

Computational complexity of PIS maintenance process depends on the structure of the domain knowledge and architecture of the system. In our experimental domain we had always only several records (say tens) in PIS and almost each of them has been implemented by different agents. Therefore any PIS revision was fast. However in many cases the state space modeled by the multi-agent system may be complicated and exhaustive. Some agents may administer substantial number of plans in their PIS and consequently the plan revision and PIS reordering within these agents may become a bottleneck of the community operation.

This problem can be addressed either on the

- meta-agent level, where meta-agent learning is used for bottleneck detection which results in cloning an identical agent that helps to process the request (we will discuss this approach later in the study)
- single agent level, where we want an agent itself to identify which plans are obsolete and unexploited and retract these from PIS.

As in the case of cooperation neighborhood optimization, single agent can parse its historical record of PIS utilization and retract unexploited plans (with respect to some t_{out}). Similarly reordering of the PIS is not required if each of the revised plans (or its part) did not change its cost more than t_{rel} in percent.

The nature of the problem indicates that for very complex problems, where some optimization of the problem solving neighborhood is inevitable, it is better to start with minimal neighborhood and build it up with experience rather than pruning the maximal neighborhood. In the latter case it may happen that agents will be so heavily overloaded from the very start that they will not make it up to some optimal problem solving neighborhood. This assertion is also subject of experimental validation.

Sub-task Assignment Optimization

So far we have been optimizing the overall communication load of the community. The other aspect that may be a subject of improvement is the quality of solution the multi-agent system finds. Let us discuss the planning system (ProPlanT is an instance). A planning system produces plans. Although each agent optimizes its sub-task decomposition, this does not mean that the system will operate optimally for a sequence of requests.

Example:

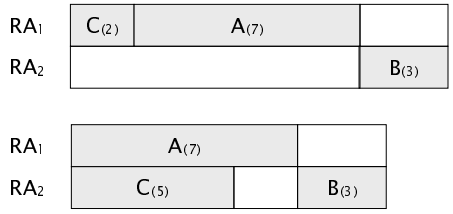


Figure 4 – Sub-task Assignment Optimization

There are three agents, one DAs and two RA. The DA is able to decompose tasks x (x = ⟨a < b⟩) and y (y = ⟨c⟩). RA_1 can do a in 7 time units and b in 2 and RA_2 can perform b in 3 and c in 5. If DA is required to plan x and y simultaneously it will plan (under assumption of

local optimality) RA_1: $c_{(0–2)}$, $a_{(2–9)}$ and RA_2: $b_{(9–12)}$, although the globally optimal would be RA_1: $a_{(0–7)}$, and RA_2: $c_{(0–5)}$, $b_{(7–10)}$

As we were analyzing past performance of the system above, we want the system to be able to predict future request in order to avoid this local optima. When assigning the subtasks for a given request, it uses the knowledge about the next request, to do this assignment in a way, that the resulting plan for the current and for the next request is optimal.

The first learning issue is to predict future events. But the problem is that even if the future requests are known, the algorithm to find the optimal assignment is very complicated. The second learning goal is to find shortcuts for this complicated algorithm. This means to find simple rules that lead mostly to the same (optimal) result, but are much more efficient. The third learning task, which arises in this setting, is to learn when to use the simple original algorithm for sub-task assignment and when to use the sophisticated algorithm. Clearly, if the plans generated by the simple algorithm do not contain any sub-optimal structures, the more complicated algorithm could make them only worse.

All learning tasks optimize the quality of the resulting plans. The respond time could on the other hand increase, because of the use of more complicated algorithms in the DA. Learning is mostly performed in a distributed way within the DA (even though it would be possible, that the MA has to provide feedback and information to them). The DA can apply their learned knowledge on their own. As this directly contradicts our effort to minimize overall problem solving complexity of the multi-agent system as whole we did not pursue any implementation of algorithms addressing tasks in this paragraph.

Number of Unachieve Requests Reduction

Un-achieve requests appear in a multi-agent system, if a DA requests sub-tasks from different agents, and at least one of these agents replies with a sorry message. In this case the agents, which already have fixed the plans for their tasks have to undo these plans, because the whole plan cannot be achieved. Naturally un-achieves are not desired, not only because they cause additional computation, but also because it can be quite complicated to undo a plan, which is already fixed.

The possibility of situations in which un-achieve is needed, arises from the fact, that the up-date of the knowledge bases takes some time. In the last example in the paragraph `'Frequency'` we commented on a case where two DAs are trying to send a request to a RA simultaneously, both believing that the RA is able to achieve the given subtask in time as advertised. But the RA can achieve this task only once, so that the request, which reaches the RA first, is answered positively, the other one negatively. In order to keep consistent with the rest of plan that relied upon the failed task, already contracted subtask has to be replanned. A task is replanned so that it is firstly canceled (by means of an `unachieved` requests) and than it is planned again (by means of an `achieve` requests)

The idea is to reduce the number of un-achieve request by learning which tasks or agents are very likely to cause them and to use this information to wait for the answer of these agents, before sending requests to other agents. On one hand, this leads to an increase of the respond-time, because the system is not used optimally. On the other hand this could reduce the respond-time and the number of messages, by reducing the number of un-achieve requests. So this kind of learning makes only sense in settings, in which un-achieve requests occur often. The learning is performed in the distributed way again.

Meta-Agent Learning

There are several patterns of the inter-agent communication and causalities among agents mutual interaction that are rather hard to detect at the level of single agent learning. Though each agent will

carry out an optimal decision, it may happen that the problem solving process will be inefficient form the global perspective of the entire community.

Bottleneck Detection

We have experimented with bottleneck detection. Our meta-agent was made to analyze the past log of communication in order to detect a bottleneck. If there is a bottleneck in the system it means that that there is an imbalance in the use of the underlying resources, in opposite to the term overload of the system, which stands for insufficiency of the whole system to deal with incoming requests.

Example:

> Having a simple community of four DA agents and one RA agent we will show very naïve bottleneck that cannot be detected on the single agent level. Let us assume that DA_1 can choose between DA_2 and DA_3, where DA_2 advertises to be less overloaded than DA_3. While DA_3 directly contracts RA agent, DA_2 contracts DA_4 who contracts RA. Let DA_4 be a bottleneck. In this community DA_4 will get heavily overloaded as DA_1 who optimizes the decomposition does not have the knowledge of DA_4 status.

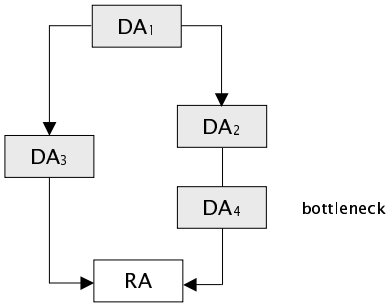


Figure 5 – Sub-task Assignment Optimization

In principle we distinguish between two types of bottlenecks. Under a communication bottleneck we will understand a bottleneck caused by communication imbalance within the community. Such a bottleneck may be caused by an agent with low computational power or more likely by slow communication links. On the other hand operational bottleneck is caused by imbalance in specialized agents' operational load. In the case of planning multi-agent system an agent with plan that causes an overall agents delay will be recognized as an operational bottleneck. When considering our example the 3bA model will preventing operational bottlenecks. If the bottleneck agent (DA_4) provides only a very slow plan, it advertises this information DA_2 who incorporates this information into its advertise to DA_1. The DA_4 will therefore prefer the plan provided by DA_2 and will thus avoid the bottleneck agent DA_4. In the following we will discuss communication bottlenecks.

Detecting and possible prediction of bottlenecks will contribute to the overall efficiency improvements of the multi-agent system operation. Firstly agents who do not have knowledge of existence of the bottleneck will replace its locally optimal decision making with a globally optimal decision making provided they acquire information about the bottleneck from the meta-agent. In the context of the previous example DA_1 would prefer DA_3 as it is aware of DA_2 participating on a bottleneck route. Secondly (and maybe more importantly) the meta-agent will have the capacity to suggest construction of the stand-in agent, an agent with same functionality and communication links as the bottleneck agent. Stand-in agent will take over some of the critical load.

When analyzing the operation of the bottleneck, the meta-agent will have to transform the log of inter-agent communication into a model of agent's operational plans. For this parsing the `achieve`

message content is sufficient, from which the agents' load will be easily computed. Attributes of training examples will be in this case loads of the community members. In the case of identifying a communication bottleneck things will get a bit more complicated. Agents load will be worked out from the number of messages received in a certain time window. For this we have to store specific receive/sent times in the log of community messages.

Conclusions

Multi-agent systems seems to be an adequate vehicle to handle information processing in complex technical systems, especially in the case these systems are geographically distributed.

The vast volume of the inter-agent communication traffic, i.e. the number of messages passed among the agents, is often considered as a weakness of multi-agent solutions. Assuring security in the systems with intensive message traffic becomes to be a hard-core problem. One of the opportunities offered by the approach proposed within the frame of this project is to significantly reduce the number of messages to be necessarily communicated. The tri-base acquaintance model of social knowledge located in the wrappers of the individual agents is used for this purpose. Both the tri-base model and the knowledge maintenance strategy (based on the *subscribe-advertise* paradigm) do enable the shifting of the substantial part of the traffic from the critical times to idle time slots of the system.

Moreover, the number of messages communicated in the idle time doesn't influence the system performance. That means that the idle-time messages can be used for security improvements (exchange of security confirmations, sending of deeply coded messages, consistency checking based on information redundancy etc.). Thus, the idle time message exchange can be used for detection of intruders or for increasing the global system security in general.

Machine learning techniques provide substantial enhancement of the tri-base model, which could help to increase the overall adaptivity and flexibility of the global system in conditions of slightly changing environment. We have analyzed the role of machine learning algorithms on

- a single agent level, where characteristics of a single agent can be improved by various self-learning procedures in order to optimize agent's cooperation neighborhood and its problem solving scope and improve efficiency of the knowledge maintenance mechanism,
- a meta-level, where the results of monitoring of the communication traffic on the level of the meta-agents are explored in order to improve overall systems efficiency by e.g. bottleneck detection.

A new idea presented in this report considers an introduction of the so called stand-in-agent which represents a functional replica (a clone) of an overloaded agent to help to solve the operational bottleneck. It looks like the technology of the stand-in-agents should be studied in more depth, as it can be used e.g. for estimating the behavior of temporarily lost/hidden agents (this problem is typical for the coalition formation domain) and to keep the overall system integrity (required for the system security purposes) more efficiently.

Rererences:

[Cao 97] Cao W., Bian C.-G., Hartvigsen G.: *Achieving Efficient Cooperation in a Multi-Agent System: The Twin-Base Modelling.* In: Cooperative Information Agents (Kandzia P., Klusch M. eds.), LNAI 1202, Springer-Verlag, Heidelberg, 1997, pp. 210-221

[Dix 00] Dix J., Subrahmanian V.S., Pick G.: *Meta Agent Programs.* Submitted to Journal of Logic Programming, to appear in 2000

[Durst 99] Durst R., Champion T., Witten B., Miller E Spanguolo L.: *Testing and Evaluating Intrusion Detection Systems.* In: Communication of ACM, July 1999, pp. 53-62

[Lamb 96] Lamb N., Preece A.: *Verification of Multi-Agent Knowledge-Based Systems*. In: Proc ECAI-96 Workshop on Validation, Verification and Refinement of KBS, August 1996.

[Marik 98] Mařík V., Pěchouček M., Hazdra T., Štěpánková O.: *ProPlanT - Multi-Agent System for production Planning*. In: Proc. of XVth European Meeting on Cybernetics and Systems Research, Vienna 1998, pp. 725-730

[Marik 99b] Mařík V., Pěchouček M., Lažanský J., Koutník J., Štěpánková O.: *Re-planning in the Multi-agent System for Production Planning*. In: Artificial Intelligence and Soft Computing. Vol.1. Calgary: IASTED, 1999, pp. 407-411 - ISBN 0-88986-251-6

[Mařík 99a] Mařík V., Pěchouček M., Lažanský J., Roche, C.: *PVS'98 Agents: Structures, Models and Production Planning Application*. In: Robotics and Autonomous Systems, vol. 27, No. 1-2, Elsevier, 1999, pp.29-44. ISSN 0921-8890

[Mitchell 97] Mitchell T.M.: *Machine Learning*. Morgan-Kaufman, 1997

[Nwana 97] Nwana H.S., Ndumu D.T.: *An Introduction to Agent Technology*. In: Software Agents and Softcomputing (Nwana H.S. and Azarmi N. eds.), LNAI 1198, Springer-Verlag, Heidelberg, 1997, pp. 3-26

[Oliviera 99] Oliviera E. - Fischer K. - Štěpánková O.: *Multi-agent Systems: Which Research for Which Applications*. In: Robotics and Autonomous Systems, vol. 27, No. 1-2, 1999, pp. 91-106. ISSN 0921-8890

[Pechoucek 00a] Pěchouček M., Mařík V., Štěpánková O.: *Role of Acquiantance Models in Agent-Based Production Planning Systems*. CIA'2000, Cooperative Information Systems Workshop, Intrnational Conference on Multi-agent Systems – ISCMAS2000, July, Boston.

[Pechoucek 00b] Pěchouček M., Mařík V., Štěpánková O.: *Potential Roles of Acquaintance Models and Meta-Agents in Communication Safety/Efficiency Improvements*, phase 1 report, Multi-Agent Systems in Communications – AFOSR research project, project contract no.: F61775-99-WE099

[Pechoucek 00c] Pěchouček M., Mařík V., Štěpánková O.: *Implementation of Tri-Base Acquaintance Model in ProPlanT Multi-Agent System*, phase 2 report, Multi-Agent Systems in Communications – AFOSR research project, project contract no.: F61775-99-WE099

[Russell 97] Russell S.J.: Rationality and intelligence, Artificial intelligence 94 (1997), pp. 57-77

[Stepankova 96] Štěpánková O., Mařík V., Lažanský J.: *Improving Cooperative Behaviour in a Multi-agent System*. In: Proc. of the IFIP World Conference on IT Tools (Terashima N., Altman E., eds.), 2-6 September 1996, Canberra, (ISBN 0 412 75560 2), Chapman and Hall 1996

[Sycara 95] Sycara K.: *Intelligent Agents and Information Retrieval*. Unicom Seminar on Intelligent Agents and their Business Applications, 8-9 November, London, 1995, pp. 143-159

[Tambe 97] Tambe M.: *Towards Flexible Teamwork*. Journal of Artificial Intelligence Research, AI Access Foundation and Morgan Kaufman Publishers, 7, pp. 83-124, 1997

[Tate 99] Tate A., Polyak S., Jarvis P.: *Knowledge Acquisition for AI Planning within the O-Plan Project*. In: Proceedings of the 1st Workshop of the PLANET Knowledge Acquisition Technical Coordination Unit, Salford University, Salford, UK, April 1999.

[Weiss 98] Weiss G. (ed.): *Distributed Artificial Intelligence Meets Machine Learning*. LNAI, Vol 1221, Springer-Verlag, 1998

[Wooldridge 95] Wooldridge M., Jennings N.: *Intelligent Agents: Theory and Practice*. The Knowledge Engineering Review, 10 (1995), No.2, pp. 115-152

[Zhong 97] Zhong N., Kakemoto Y., Ohsuga S.: *An Organised Society of Autonomous Discovery Agents*. In: Cooperative Information Agents (Kandzia P., Klusch M. eds.), LNAI 1202, Springer-Verlag, Heidelberg, 1997, pp.183-194

Appendix 1 – Tri-base Acquaintance Model

Each of the tri-base agents is equipped with a specific knowledge of behavior of the collaborating agents encoded in the *tri-base acquaintance model* [Mařík 98] in order to provide an optimal task decomposition. Prior to formalising the model let us introduce several primitives we will use throughout the course of explanation. Let Θ be a set of all agents within the community and S a set of all tasks the community members are able to decompose. For each $\mathcal{A} \in \Theta$ let

- $\alpha(\mathcal{A}) \subseteq \Theta$ be an agent's *total neighbourhood*, a set of agents an agent $\mathcal{A}$ is aware of,
- $\beta(\mathcal{A}) \subseteq \mathsf{S}$ be the set of all tasks the agent $\mathcal{A}$ is able to decompose,
- $\gamma(T)$, contains all possible plans for decomposing the task $T \in \mathsf{S}$. Plan for the task T is in the form $\langle T, S, \mathscr{O}, \mathscr{E} \rangle$, where S is a set of subtasks which ensure completion of the task T provided that their processing meets precedence constraints $\mathcal{O}$ and applicability constraints $\mathcal{C}$.
- $\omega(\mathcal{A},T) \subseteq \gamma(T)$ contains those plans for the task T an agent $\mathcal{A}$ knows about (if $T \notin \beta(\mathcal{A})$ then $\omega(\mathcal{A},T) = \emptyset$).

The following sets provide time dependent information. Let

- $\varepsilon^t(\mathcal{A}) \subseteq \alpha(\mathcal{A})$ be the agent's current *cooperation neighbourhood* , a set of agent's $\mathcal{A}$ collaborators at the time instant t,
- $\tau^t(\mathcal{A}) \subseteq \beta(\mathcal{A})$ contain the tasks being solved by the agent $\mathcal{A}$ in a time instance t and the set
- $\pi^t(\mathcal{A}) \subseteq \beta(\mathcal{A})$ be a collection of tasks an agent $\mathcal{A}$ is supposed to have pre-prepared in advance0 in time instance t

Within the tri-base model each agent maintains three knowledge bases where all the relevant information about the rest of the community is stored. We distinguish among:

CO-OPERATOR BASE (CB) – maintains permanent information on co-operating agents (i.e.: their address, communication language, and their predefined responsibility). This type of knowledge is expected not to be changed very often. $\mathrm{CB}(\mathcal{A})$ is then defined as

$$\mathsf{CB}(\mathcal{A}) \stackrel{\text{def}}{=} \{\langle B, \mathsf{Addr}(B), \mathsf{Lang}(B), \beta(B)\rangle\}_{B \in \alpha(\mathcal{A})} \text{ where}$$

$\mathsf{Addr}(B)$ specifies agent's the address, $\mathsf{Lang}(B)$ language it communicates, as already mentioned $\beta(B)$ is a set of tasks the agent accounts for and the set $\alpha(\mathcal{A})$ denotes members of the agent's $\mathcal{A}$ scope of the community.

TASK BASE (TB) – stores in its *problem section* (PRS) general problem solving knowledge – (i) information on possible decompositions of the tasks to be solved by the agent and (ii) in its *plan section* (PLS) it maintains the actual and most up-to-date plans on how to carry out those tasks, which are the most frequently delegated to the agent - the owner of the task base, those denoted as $\pi^t(\mathcal{A})$. Formal definition of the $\mathrm{TB}(\mathcal{A})$ is then

$$\mathsf{TB}(\mathcal{A}) \stackrel{\text{def}}{=} \langle \mathsf{PRS}(\mathcal{A}), \mathsf{PLS}(\mathcal{A})\rangle, \text{ where}$$

$$\mathsf{PRS}(\mathcal{A}) \stackrel{\text{def}}{=} \bigcup_{T \in \beta(\mathcal{A})} \omega(T,\mathcal{A}) \text{ and}$$

$$\mathsf{PLS}^t(\mathcal{A}) \stackrel{\text{def}}{=} \{\langle T, \langle\{\langle s, B\rangle\}_{s\in S}, \mathscr{O}, \mathscr{C}, \mathsf{Trust}(T)\rangle\rangle\}_{T\in \pi^t(\mathcal{A})},$$

where for any $\langle T, \langle\{\langle s, B\rangle\}_{s\in S}, \mathscr{O}, \mathscr{C}, \mathsf{Trust}(T)\rangle\rangle \in \mathsf{PLS}^t(\mathcal{A})$ where exist $\mathscr{O}_1$, $\mathscr{C}_1$ such that following constraints are met $\langle T, S, \mathscr{O}_1, \mathscr{C}_1\rangle \in \mathsf{PRS}(\mathcal{A})$, $B \in \varepsilon^t(\mathcal{A})$, $s \in \beta(B)$ and as is a specialisation of $\mathscr{C}_1$ reflecting the considered allocation of the asks $s \in S$, $\mathscr{O}$ is refinement of $\mathscr{O}_1$ and both $\mathscr{O}$ and $\mathscr{C}$ are valid.

STATE BASE (SB) –stores in its *agent section* (AS) all information on current load of co-operating agents. This part of the state base is updated frequently and informs the agent who is busy and who is available for collaboration. In the *task section* (TS) there is stored information on status of tasks the agent is currently solving. Formal description of the $\mathrm{SB}(\mathcal{A})$ of agent $\mathcal{A}$ is thus

$$\mathrm{SB}(\mathcal{A}) \stackrel{\text{def}}{=} \langle \mathsf{AS}(\mathcal{A}), \mathsf{TS}(\mathcal{A})\rangle, \text{ where}$$

$$\mathsf{AS}(\mathcal{A}) \stackrel{\text{def}}{=} \{\langle B, \mathsf{Cap}(B), \mathsf{Load}(B), \mathsf{Trust}(B)\rangle\}_{B\in \varepsilon^t(\mathcal{A})} \text{ and}$$

provided that agent's B capability has the form of $\mathsf{Cap}(B) \equiv \{\langle T, \mathsf{Cost}(T)\rangle\}_{T\in \beta(B)}$, overall agent load is $\mathsf{Load}(B)$, and trust in this information in $\mathsf{Trust}(B)$.

$\mathsf{TS}(\mathcal{A})$ contains relevant information on all the tasks agent $\mathcal{A}$ agreed to supervise recently. This set is denoted by $\tau^t(\mathcal{A})$. Formally

$$\mathsf{TS}(\mathcal{A}) \stackrel{\text{def}}{=} \{\langle T, \mathsf{Dec}(T), \mathsf{State}(T), \mathsf{Trust}(T)\rangle\}_{T\in \tau^t(\mathcal{A})},$$

where decomposition $\mathsf{Dec}(T)$ is taken from the $\mathsf{PLS}^{t_1}(\mathcal{A})$ at the moment of contract (time t_1). $\mathsf{State}(T)$ partitions subtasks from $\mathsf{Dec}(T)$ into three parts: subtasks finished, actually processed, and the rest. The record is complemented with the trust value $\mathsf{Trust}(T)$ denoting trust in the plan of the task T.

The agent is supposed to select an optimal plan from the $\mathsf{PLS}^t(\mathcal{A})$, where an appropriate amount of plans prepared in advance is stored. By this it does not need to contract peer agents in order to find out the most appropriate (optimal) offers for further problem delegation. Knowledge stored in the PlS will help the agent to decide by itself. It is obvious that limiting the communication among agents will in its own way decrease the computational complexity of the entire problem. The price we have to pay for this, is a communication increase among agents when updating the SB.

The model maintenance algorithms are based on a simple subscribe/advertise mechanism. After parsing the PrS knowledge, each agent identifies possible collaborators and subscribes these for reporting on their statuses. The subscribed agent advertises its load, capabilities and task completion times and costs estimates either periodically or when either of these changes. This mechanism

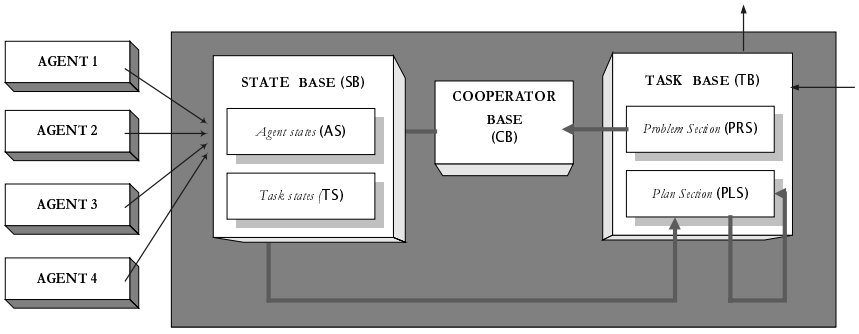


Figure 6 – Tri-base Model Planning

facilitates the agent to make the best decision with no further communication.

The communication savings of this approach may be seen as a specific communication load shift. The communication load is minimised in the agent's *critical time* (i.e.: moment when the agents are required to fulfil, through delegation, a request) while a new advertising activity appears in the agent's *idle time*. Moreover each request is answered quickly but it brings substantial communication flows following the request fulfilment. The truth is that successful operation of such a multi-agent system depends on the community lifecycle. In order to utilise this acquaintance model based mechanism and to guarantee its communication savings for the frequency f of requirements to plan the following condition should be met

$$1/f \geq t_c + t_i,$$

where t_c is maximum amount of time spent in the *critical time* planning and t_i is maximum amount of time needed for processing the subscription/advertise mechanism in agents' *idle time*.

There are two issues to be addressed here: (i) how many peer agents to subscribe and (ii) how many plans to keep pre-prepared. *Cooperation neighbourhood* $\varepsilon^t(A)$ denotes a collection of such agents B who belong to agent's A scope of collaboration. All agents $B \in \varepsilon^t(A)$ are subscribed by the agent A. Maximum neighbourhood $\varepsilon^t(A)$ is specified when the agent parses its problem solving knowledge-base (PRS) and detects who will be needed for further collaboration. $\varepsilon^t(A)$ is fixed here for the entire course of agent's decision making. In this way, the agent is collecting lots of redundant data, which slows the community down substantially. The possible role of the meta-agent can be in analysing inter-agent communication and optimising the $\varepsilon^t(A)$ neighbourhood.

On the other hand, $\pi^t(A)$ neighbourhood, agent's A scope of reasoning, specifies how many of plans from PRS will be kept instantiated and on-line evaluated in PLS of the TB. We distinguish among two marginal cases. If $\pi^t(A) = \emptyset$, there is no plan pre-prepared and planning in the critical time requires substantially more of computational time for constructing the PLS plans. If $\pi^t(A)$ is maximal, there is no time needed for constructing the PLS plans, but each minor change within community results in massive re-computation and re-evaluation of PLS plans. Designers of the system rely on meta-agent machine learning capabilities in setting $\pi^t(A)$ agent's scope of reasoning.

Appendix 2 – Tri-base Agent Implementation

This appendix describes the tri-base agent implementation. The tri-base agent consists of two parts, a wrapper and a body. While the body is a computational instantiation of the tri-base acquaintance model that was designed within the first and the second phases of project and tested in the third phase of the project, the wrapper has been developed in order allow encapsulation of the tro-base agent within the ProPlanT multi-agent system. The wrapper translates messages from the body to the low-level communication format and vice-versa. The wrapper and the tri-base body are written in Java language.

Firstly we will describe implementation of the Java ProPlanT wrapper and than we will present and comment an object model of the tri-base body (in figures referred to as a pma agent).

ProPlanT Communication

The agent (wrapper) must communicate in TCP-IP protocol. Every agent is uniquely identified by the IP address and port. Every agent has also its symbolic name.

Messages that are sent between agents can be in two format, control and KQML. Control messages allow technical around for community (born and dead of agents). KQML messages are used to body of agent. *Figure 7* describe, which messages are sent between levels. Notice, that body of agent can process only KQML messages and also can sent only KQML messages again.

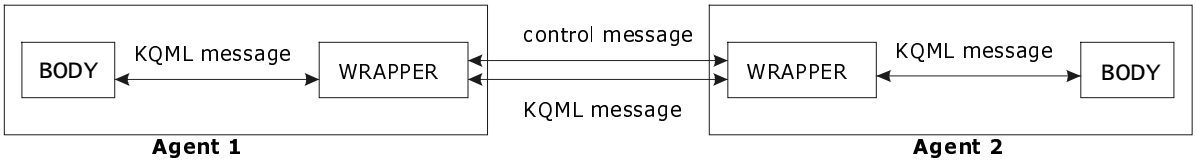


Figure 7 - Communication schema between two agents

How to write your own agent

In order to build your own agent we recommended to extend class **`proplant.wrapper.Wrapper`** and override its method **`processMessage(KQMLMessage kqmlMessage)`**

You can also extend class **`proplant.Agent`** and override methods that are discerned by performative.

- **`processSubscribe(KQMLMessage)`**
- **`processRegister(KQMLMessage)`**
- **`processUnRegister(KQMLMessage)`**
- **`processAchieve(KQMLMessage)`**
- **`processUnAchieve(KQMLMessage)`**
- **`processAdvertise(KQMLMessage)`**
- **`processEvaluate(KQMLMessage)`**
- **`processSorry(KQMLMessage)`**

- **processReply(KQMLMessage)**
- **processReady(KQMLMessage)**

But if you need to process some other performs, you must override method **processMessage(KQMLMessage)**.

Processing results in a new KQML message that must be sent to some other agent. You must use method **sendMessage(String agentName, KQMLMessage kqmlMessage)**. If you need to send a message to all agents in community, you can use method **sendBroadcastMessage(KQMLMessage)**.

For demonstration example can look at **MyAgent.java**.

Installation and Compilation of Java Sources

If you want to create programs in Java, you need a Java Development Kit 1.2.2, which you can download from **http://java.sun.com/j2se**. If you don't know, which JDK you have already installed, execute from command line **c:>java -version.**

Compilation of your programs is easy. Execute from the command line **c:>javac myclass.java**. This create file **myclass.class** that can be executed as **c:>java myclass**. Because is user-unfriendly to write your code in the notepad and compile it from the command line, you can use **Borland JBuilder** or **Forte**.

Wrapper

The Wrapper is consisted of five packages.

- **proplant**
- **proplant.util**
- **proplant.wrapper**
- **proplant.messages.kqml**
- **proplant.messages.control**

Most important is the class **proplant.wrapper.Wrapper**. This class must extend the body of every agent.

Package proplant

Package proplant contains auxiliary classes

- **proplant.TaskName** – property taskName with it's set-get methods
- **proplant.AgentName** – property agentName with it's set-get methods
- **proplant.TaskAgent** – property taskName and agentName

- **`proplant.Agent`** – extended by proplant.Wrapper – an empty agent

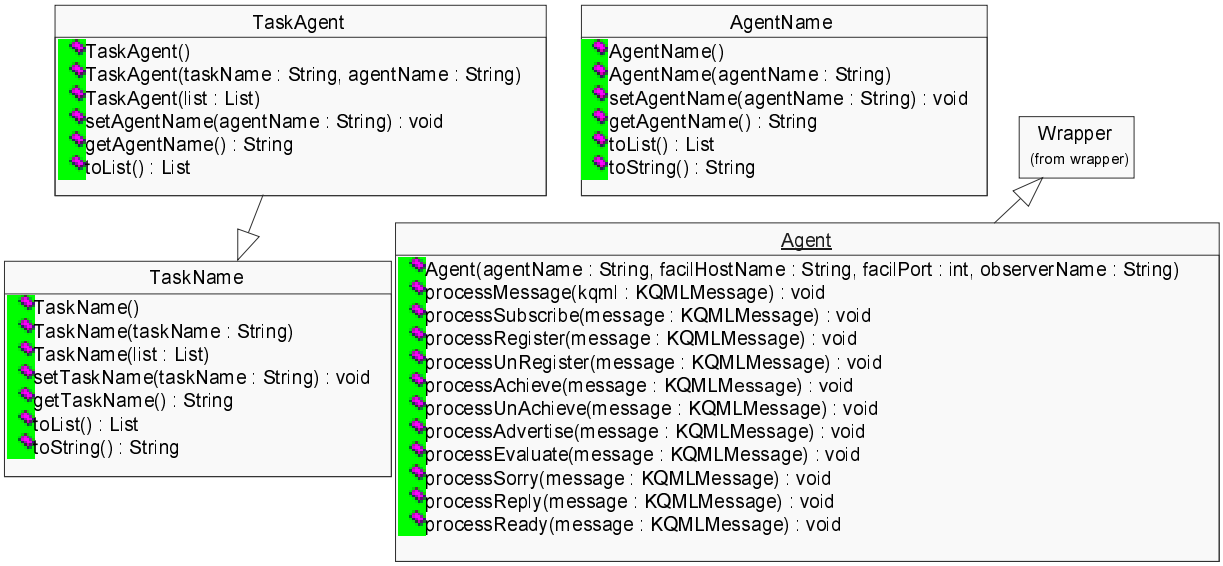


Figure 8 - Class diagram of package proplant

Package proplant.messages.kqml

This package contains four classes. The most important is the class **`proplant.messages.kqml.KQMLMessage`**. This class provides parser and builder of KQML message. Class **`proplant.messages.kqml.Content`** provides parser and builder of the content KQML message. It is supposed that content is in lisp list format. Class **`proplant.message.kqml.Trust`** provides elementary information included in the content.

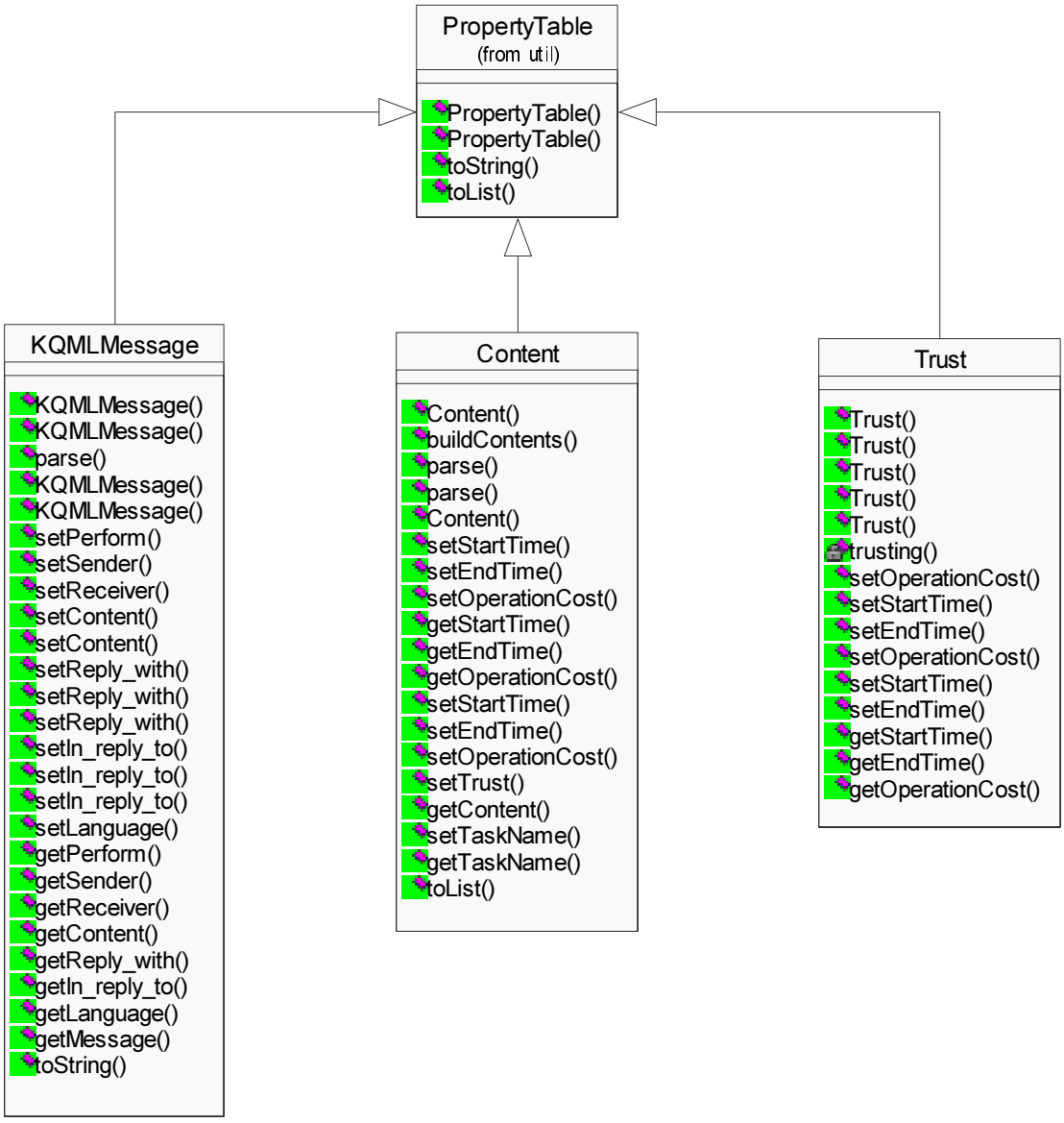


Figure 9 – Class hierarchy of package proplant.messages.kqml

Package proplant.messages.control

This package contains class **proplant.messages.control.ControlMessage** that provide parser and builder of control message. Type of command is adjustment by the class **proplant.messages.control.Commands** and type of message by the class **proplant.message.control.Types**.

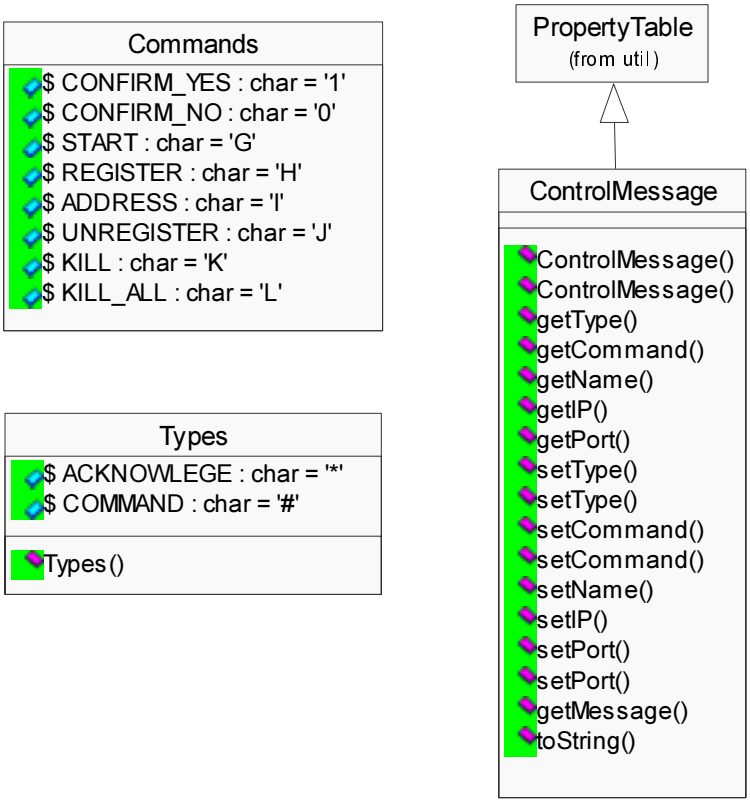


Figure 10 – Class diagram of the package proplant.messages.control

Package proplant.util

This package contains only one class **proplant.util.PropertyTable** that is used as a parent of another classes.

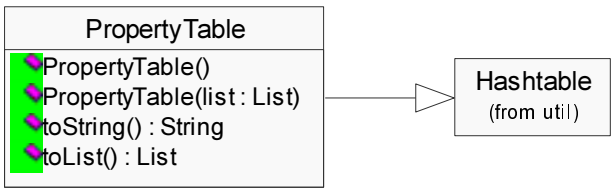


Figure 11 - Class diagram of the package proplant.util

Package proplant.wrapper

The class **proplant.wrapper.Wrapper** uses the class **proplant.wrapper.Console** as a list of messages. Wrapper writes into the console input and output KQML messages. User can later analyze this log. The class **proplant.wrapper.PhoneBook** is a list of **proplant.wrapper.AgentInfo**. It is used for translating network addresses to agent names and vice-versa. In property observerName there is stored name of agent, to which copies of outgoing messages are sent.

The basic building block of the wrapper is in the method **run**. The thread is blocked until some socket request is fired. If incoming message is in the control format, than the wrapper process it by private method **processControlMessage(KQMLMessage)**. If incoming message is in KQML format, the wrapper writes this message into an input queue and blocks itself until next request is fired. Another thread InputQueue reads messages from the input queue and fires method **processMessage(KQMLMessage)**. So messages are processed asynchronously. When agent call method **sendMessage(String agentName, KQMLMessage kqmlMessage)**, new instance of OutputThread is created, that send message throw network. Every request for sending message creates new thread. So all outgoing messages are sent asynchronously.

If you want to listens to some events e.g. for visualization, you must create class that implements **proplant.wrapper.WrapperListener** interface and connect it to wrapper by **setWrapperListener(WrapperListener)**.

New wrapper is create e.g

```
Wrapper wrapper = new Wrapper("agentName", "localhost", 1000);
wrapper.register();
wrapper.start();
```

Method **wrapper.harakiri()** unregister and destroy itself.

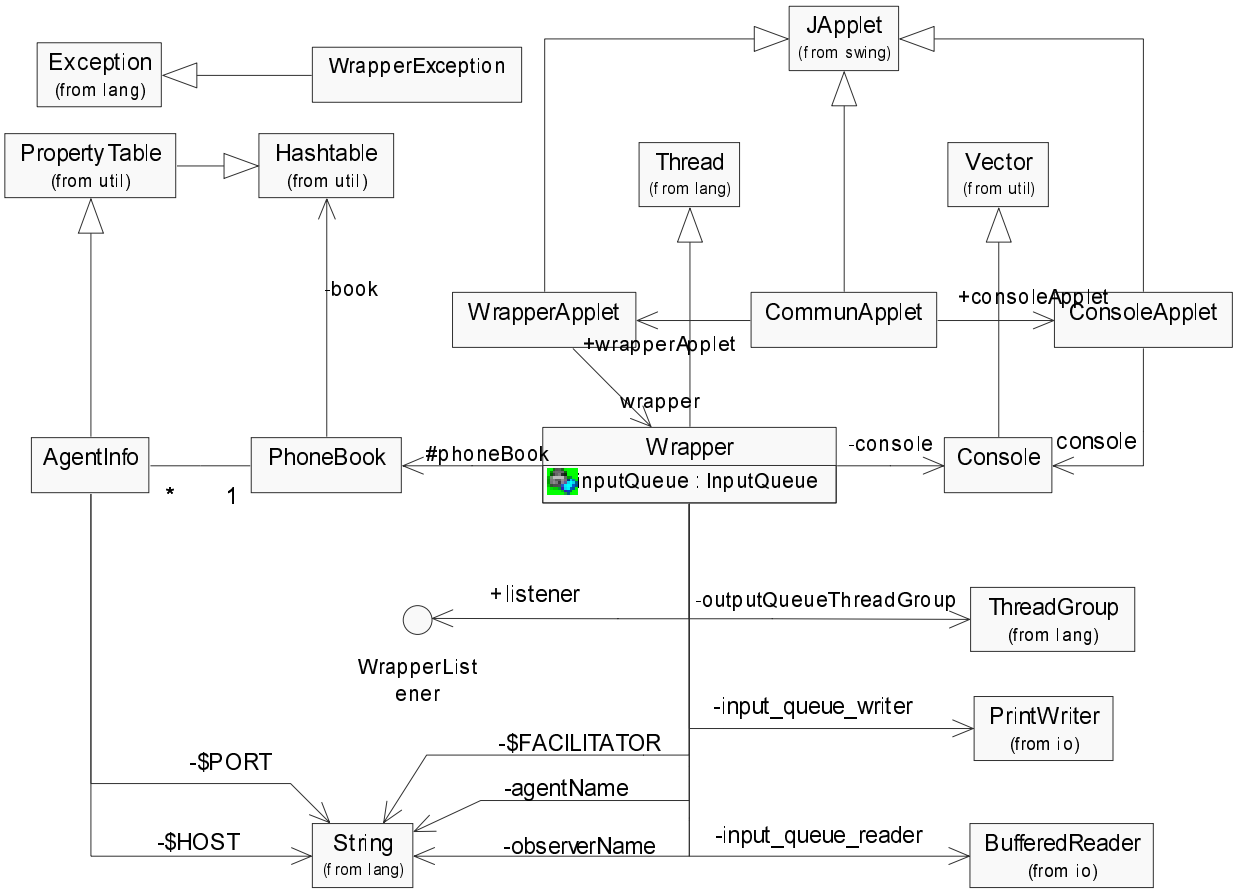


Figure 12 – Class diagram of package proplant.wrapper

Recomendation

In any processXXX(KQMLMessage message) method you need often to get content from KQMLMessage. Mehod **`message.getContent()`** returns content as String, but you need it in KIF (lisp) format. If you assume, that the content contains only one list, you can use static method **`Content.parse(String)`**.

```
Content content = Content.parse(message.getContent());
```

If you assume, that content contain more then one list, you must use static method **`Content.buildContents(String)`** that return Enumeration of contents.

```
Enumeration enum = Content.buildContent(message.getContent());
while (enum.hasMoreElements()) {
        Content content = (Content)enum.nextElement();
        ...
}
```

If you have an instance of class **proplant.messages.kqml.Content**, you get all properties by method getXXX();

```
String taskName              = content.getTaskName();
long operationCost     = content.getOperationCost();
long startTime         = content.getStartTime();
long endTime           = content.getEndTime();
```

If you need to create new content, you can use setXXX(value) methods

```
Content content = new Content();
content.setTaskName("car");
content.setStartTime(100);
content.setEndTime(200);
content.setOperationCost(300);
```

If you need to create and send KQMLMessage, you can use setXXX(value) methods

```
KQMLMessage kqmlMessage = new KQMLMessage();
kqmlMessage.setPerform(Performs.EVALUATE);
kqmlMessage.setContent(content);
kqmlMessage.setReply_with(13);
sendMessage("dealer", kqmlMessage);
```

Tri base agent

Tri base agent body consists of the task base, the state base and the cooperation base. In the task base is stored static knowlede (task section) and plans (plan section). State base consists of states of another agents (agent states) and of list of tasks, which were achieved (achieves). Cooperation base contains information about agents, which are cooperating with the agent (subscribers).

Tri base agent is packaged in package proplant.pma. Documentation you can browse here. Every base has its own package – proplant.pma.taskbase, proplant.pma.coopbase and proplant.pma.statebase. Task base is consists of two parts – proplant.pma.taskbase.tasksection and proplant.pma.taskbase.plansection. Also state base has two parts – proplant.pma.statebase.agentstates and proplant.pma.statebase.achieves.

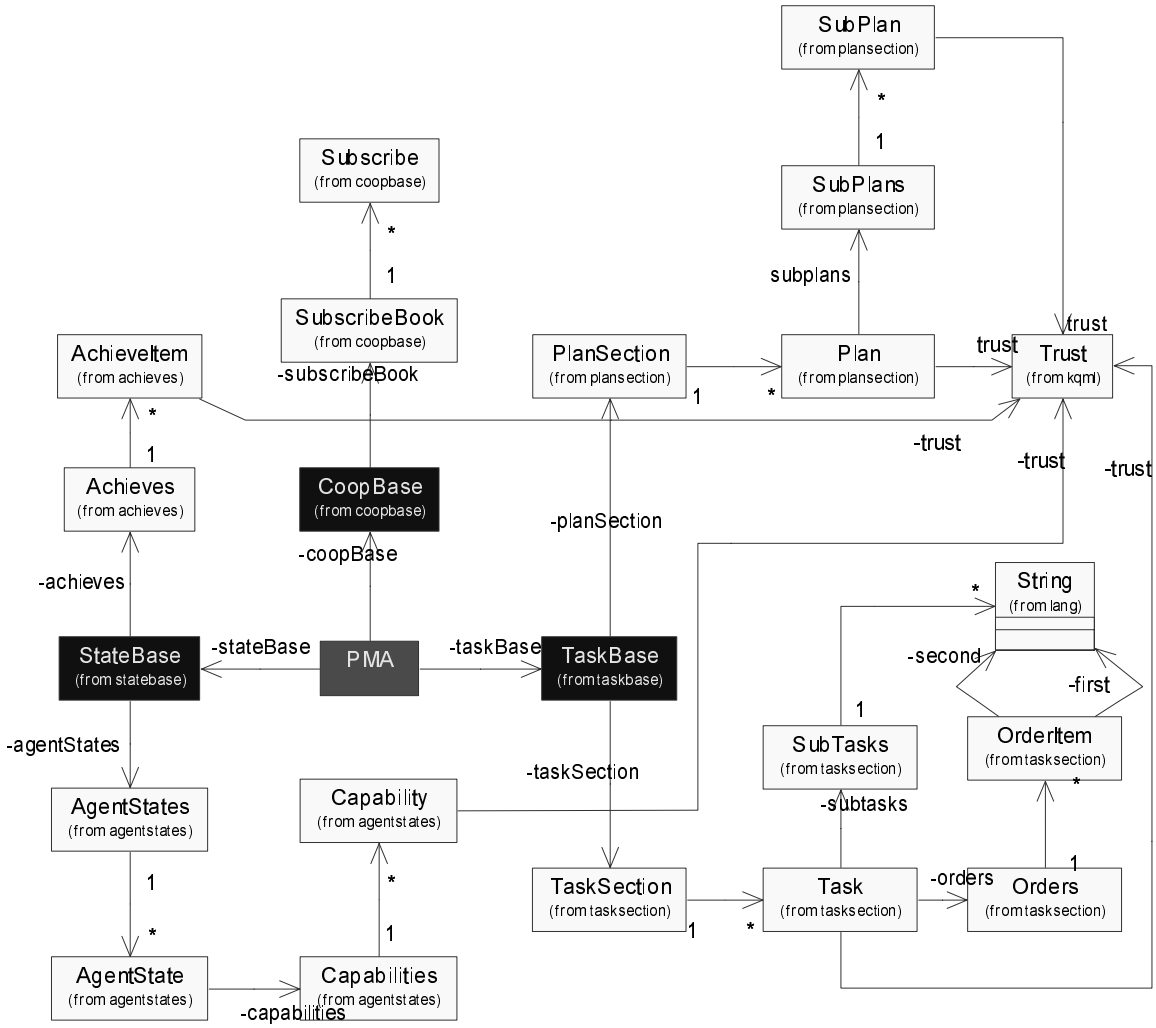


Figure 13 – Class diagram of package proplant.pma

Task base in XML format

The PMA has its knowledge stored in the taskbase file. DTD description is in file proplant.dtd. The task base file contains list of xml files. Every xml file descripts one task.

Example: management.tb and k_port.xml

```
<?xml version="1.0"?>
<!DOCTYPE Task SYSTEM "../../proplant.dtd">
<Task>
        <TaskName> compres_port </TaskName>
        <SubTasks>
```

```
            <TaskName> material_c_p </TaskName>
            <TaskName> production_c_p </TaskName>
            <TaskName> dispatching_c_p </TaskName>
        </SubTasks>
        <Orders>
            <OrderItem>
                <FirstTask> material_c_p </FirstTask>
                <SecondTask> production_c_p </SecondTask>
            </OrderItem>
            <OrderItem>
                <FirstTask> production_c_p </FirstTask>
                <SecondTask> dispatching_c_p </SecondTask>
            </OrderItem>
        </Orders>
</Task>
```

Appendix 3 – Tri-base Agent Demonstration

This appendix gives guidelines on how to run and reconstruct experiments described in the report.

If you don't have JVM installed, download it from **http://jsp2.java.sun.com/j2se/1.3/jre**

Step 1:

change directiory to **Proplant/Demo**

run **facil.exe**

Step 2:

run **agentfactory.exe** and in menu fire CreateAgents.

Step 3:

Find the agent **Bubik** (Java agent factory with coffee icon).

Make **Bubik** agent to send message to **Management** agent. Fill tab with parameters bellow.

```
Message: perform    = ACHIEVE
       reciever = Management
       content =(          (compres_port)
                           (start-time 0)
                           (end-time 1000)
                           (operation-cost 1000))
       reply with = 13
```

in **Observer** agent you can see community structure. In a few seconds **Management** sends back to **Bubik** message with perform **reply** (this you can see in Bubiks console).

Step 4:

In Observer you can see community structure and links with in-out messages. In a few time Management send back to Bubik message with perform REPLY (this you can see in Bubik's console tab).

It shows as

```
in:
(REPLY
:SENDER Management
:RECEIVER Bubik
:CONTENT ((compres_port)
          (end-time 29)
          (start-time 0)
          (operation-cost 322))
:REPLY-WITH 4
:IN-REPLY-TO 13
:LANGUAGE KIF)
```

Also agent Management has info agout this transaction. Click on agent Management - tab State base - subtab Task states.

Property Status informs, if the agent received/sent REPLY message.

Step 5:

Close facilitator - this close all community.

How run java agents

Empty agent (only wrapper)

run java -cp proplant.zip proplant.wrapper.Wrapper agentName facilitatorHostName facilitatorPort

example: java -cp proplant.zip proplant.wrapper.Wrapper Smith localhost 1000

PMA agent (tri-base)

run java -cp proplant.zip proplant.pma.PMA agentName taskBaseFile facilitatorHostName facilitatorPort

example: java -cp proplant.zip proplant.pma.PMA Management Pma/Management.tb localhost 1000

Broadcast agent

run java -cp proplant.zip proplant.broadcast.BroadcastAgent taskBaseFile facilitatorHostName facilitatorPort

example: java -cp proplant.zip proplant.broadcast.BroadcastAgent Management Pma/Management.tb localhost 1000

Factory

some agent run on the same VM

run java -cp proplant.zip proplant.factory.Factory factoryConfigFile facilitatorHostName facilitatorPort

factoryConfigFile = xml config file

example: java -cp proplant.zip proplant.factory.Factory factory.xml localhost 1000

WRITING AGENTS IN JAVA

Description

Agent in multi-agents system has generaly two parts, wrapper and core. Wraper translate messages from core to low-level comunication format and vice-versa. This white paper describe wrapper for proplant agents written in Java.

Requirements

Agent (wrapper) must comunicate by TCP-IP protokol. So every agnet is unambiguous intended by his IP address and port. Every agent has also his name that is assigned to his network address.

Messages that are sent between agents are in two format, control and KQML. Control messages allow technical upcountry for comunity (born and dead of agents). KQML messages are intended to core of agent.

Picture 1- Communication schema between two agents

Wrapper and empty core are written in Java language. Picture 1 describe, which messages are sent between levels. Notice, that core of agent can process only KQML messages and also can sent only KQML messages again.

How write my own agent

Recommendation way, how build your own agent is extend class `proplant.wrapper.Wrapper` and override its method `processMessage(KQMLMessage kqmlMessage)`

You can also extend class `proplant.Agent` and override methods that are discerned by perform

- `processSubscribe(KQMLMessage)`
- `processRegister(KQMLMessage)`
- `processUnRegister(KQMLMessage)`
- `processAchieve(KQMLMessage)`
- `processUnAchieve(KQMLMessage)`
- `processAdvertise(KQMLMessage)`
- `processEvaluate(KQMLMessage)`
- `processSorry(KQMLMessage)`
- `processReply(KQMLMessage)`
- `processReady(KQMLMessage)`

But if you need process some other performs, you must override method `processMessage(KQMLMessage)`.

As affect of processing is often new KQML message that must be sent to some other agent. For this account you have no choice. You must use method `sendMessage(String agentName, KQMLMessage kqmlMessage)`. If you need send message to all agents in community, you can use method `sendBroadcastMessage(KQMLMessage)`.

Demonstration example you can look throw at MyAgent.java.

Documentation

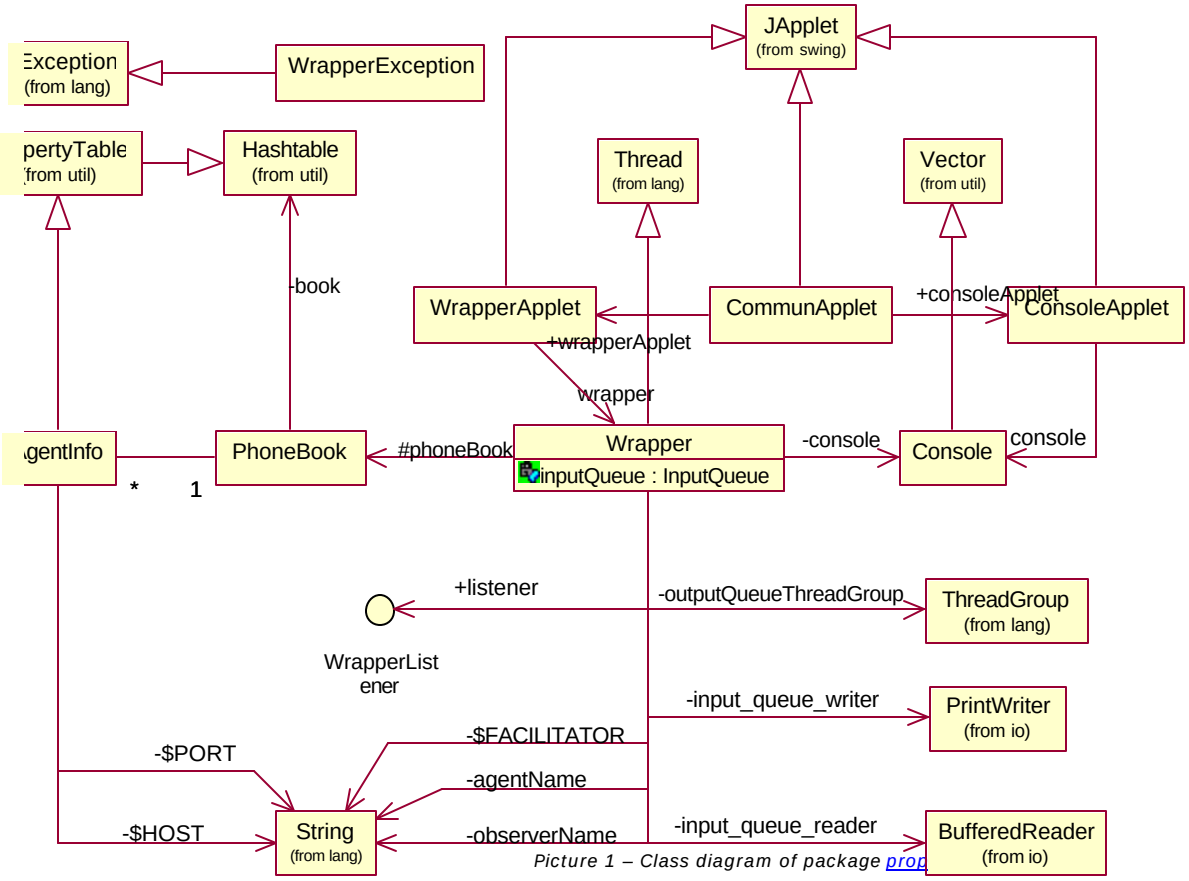
All code codumentation you can browse here.

Installation and compilation java sources

If you want create programs in java, you need Java Development Kit 1.2.2, which you can download from http://java.sun.com/j2se. If you don't know, which JDK you have already installed, try execute from comand line `c:>java -version`.

Compilation your programs are easy. Execute from command line `c:>javac myclass.java`. This create file myclass.class that can be exetuted as `c:>java myclass`. Becase is user unfriendly write your code in notepad and compile it from comand line, you can use Borland JBuilder, which is free and you can download it from http://www.borland.com/downloads.

Wrapper detailed



Picture 1 – Class diagram of package prop

Class `proplant.wrapper.Wrapper` use class `proplant.wrapper.Console` as a list of messages. Wrapper writes into console input and output messages. User can later analyze this log. Class `proplant.wrapper.PhoneBook` is list of `proplant.wrapper.AgentInfo`. It is

used for translating network addresses to agent names and vice-versa. In property observerName is stored name of agent, to which are sent copies of outgoing messages.

Base of wrapper is in method run. Thread is blocked until some socket request is fired. If incoming message is in control format, than wrapper process it by method private method processControlMessage(KQMLMessage). If incoming message is in KQML format, wrapper write this message into input queue and blockself until next request is fired. Another thread InputQueue reads messages from input queue and fires method processMessage(KQMLMessage). So messages are processed asynchronously from accepting requests. When agent call method sendMessage(String agentName, KQMLMessage kqmlMessage), is create new instance of OutputThread that send message throw network. Every request for sending message creates new thread. So all outgoing messages are sent asynchronously.

If you want listens some evens e.g. for visualization, you must create class that implements proplant.wrapper.WrapperListener interface and connect it to wrapper by setWrapperListener(WrapperListener).

New wrapper is create e.g

```
Wrapper wrapper = new Wrapper("agentName", "localhost", 1000);
wrapper.register();
wrapper.start();
```

Method wrapper.harakiri() unregister and destroy self.

Recomendation

In any processXXX(KQMLMessage message) method you need often get content from KQMLMessage. Mehod message.getContent() return content as String, but you need it in KIF (lisp) format. If you assume, that content contain only one list, you can use static method Content.parse(String).

```
Content content = Content.parse(message.getContent());
```

If you assume, that content contain more then one list, you must use static method Content.buildContents(String) that return Enumeration of contents.

```
Enumeration enum = Content.buildContent(message.getContent());
while (enum.hasMoreElements()) {
      Content content = (Content)enum.nextElement();
      ...
}
```

If you have at last instance of class proplant.messages.kqml.Content, you get all properties by method getXXX();

```
String taskName         = content.getTaskName();
long operationCost      = content.getOperationCost();
long startTime          = content.getStartTime();
long endTime            = content.getEndTime();
```

If you need create new content, you can usesetXXX(value) methods

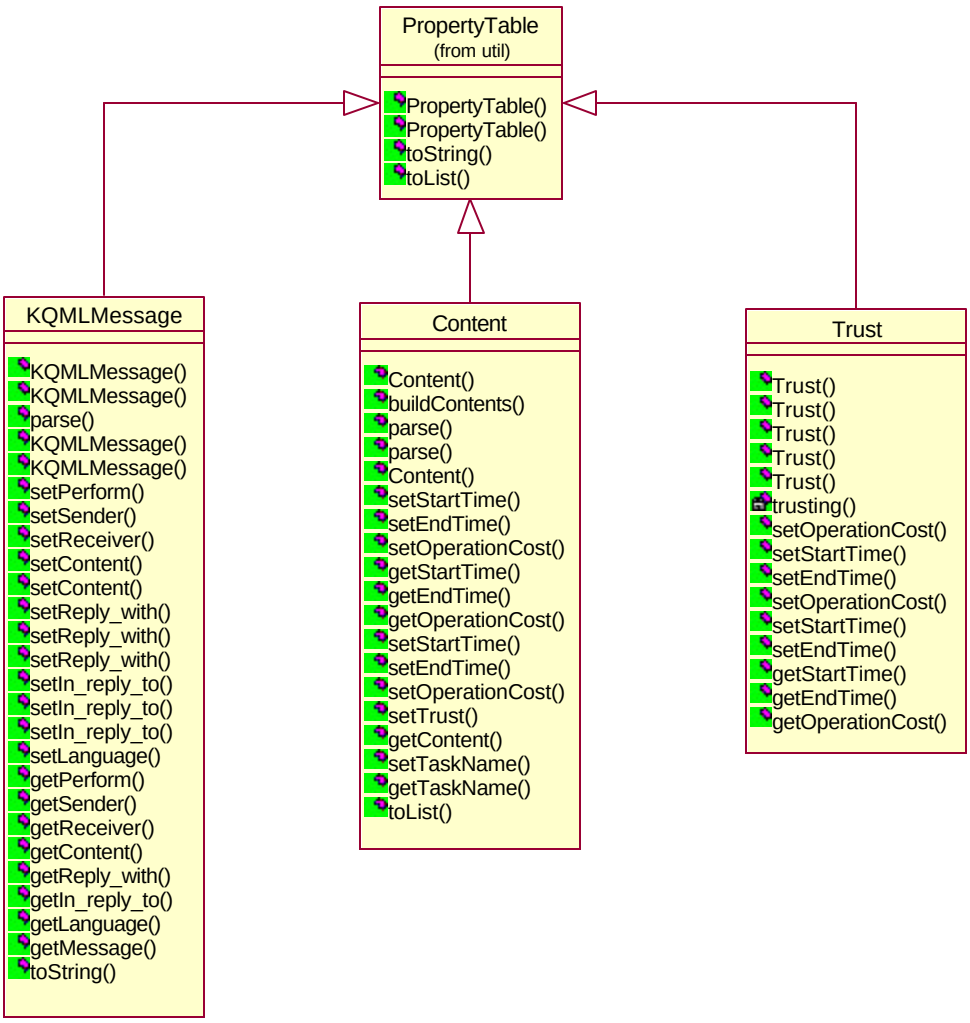
```
Content content = new Content();
```

```
content.setTaskName(“car”);
content.setStartTime(100);
content.setEndTime(200);
content.setOperationCost(300);
```

If you need create and send KQMLMessage, you can use setXXX(value) methods

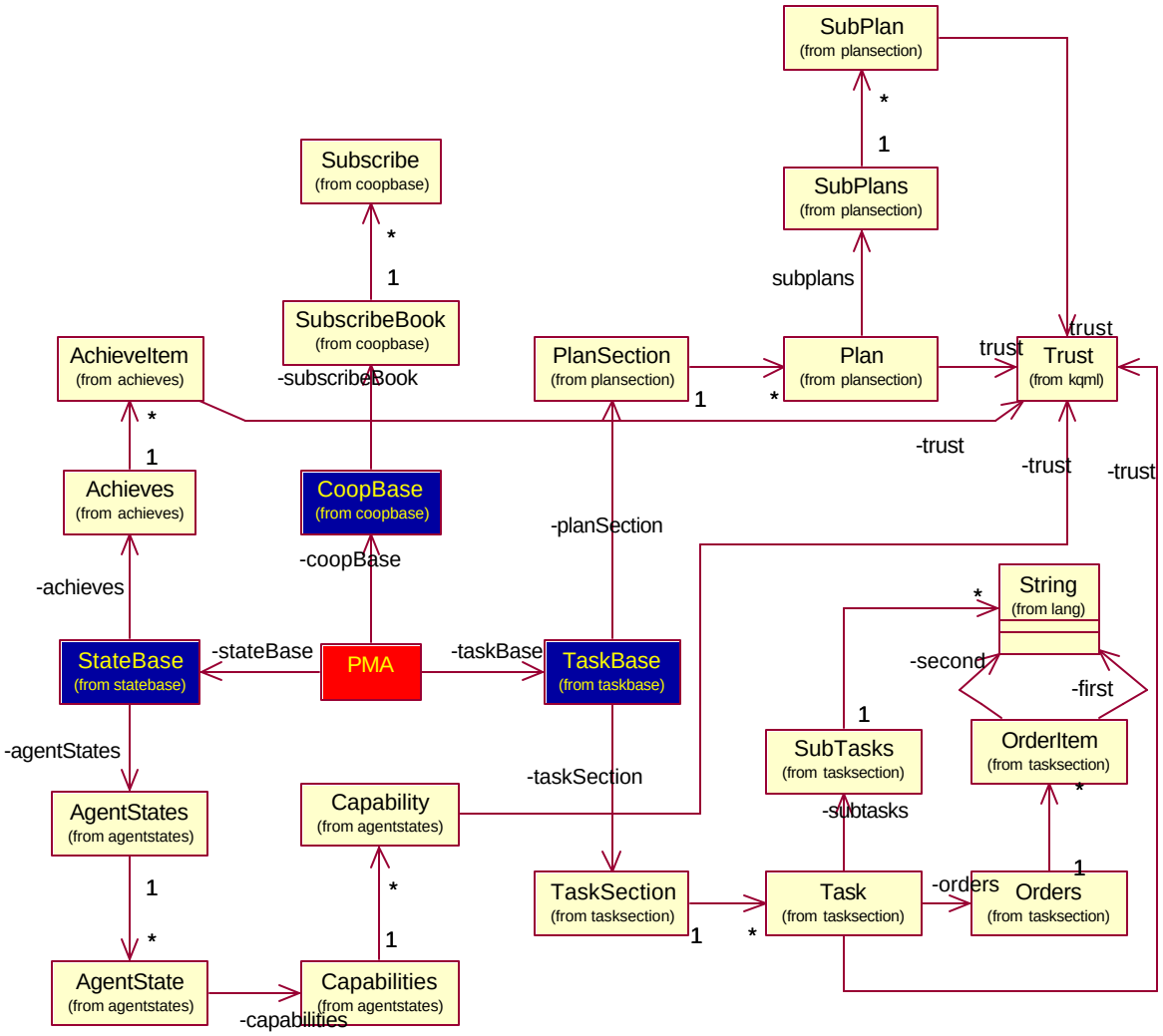
```
KQMLMessage kqmlMessage = new KQMLMessage();
kqmlMessage.setPerform(Performs.EVALUATE);
kqmlMessage.setContent(content);
kqmlMessage.setReply_with(13);
sendMessage(“dealer”, kqmlMessage);
```

KQML package



Picture 2 – Class hierarchy of package proplant.messages.kqml

PMA agent



Picture 3 – PMA agent association diagram